Swift: OpenStack Object Storage

Marica Antonacci - INFN Bari

Scuola di Cloud Computing
Bari, 27-30 Aprile 2015

Outline

- Introduction
- Key Elements & Concepts
- Architecture
- Geographically distributed cluster
- Monitoring
- Installation & Configuration

Swift

- Swift is the software behind the OpenStack Object Storage service.
- Written in python. Over 100+ contributors from many Org.
- Provides a simple storage service for applications using RESTful interfaces
- Provides maximum data availability and storage capacity.

Production deployments

- Wikipedia
- Rackspace
- Hp Cloud
- MercadoLibre
- Disney
- …

Swift Key Elements

Massive Scaling with Eventual Consistency

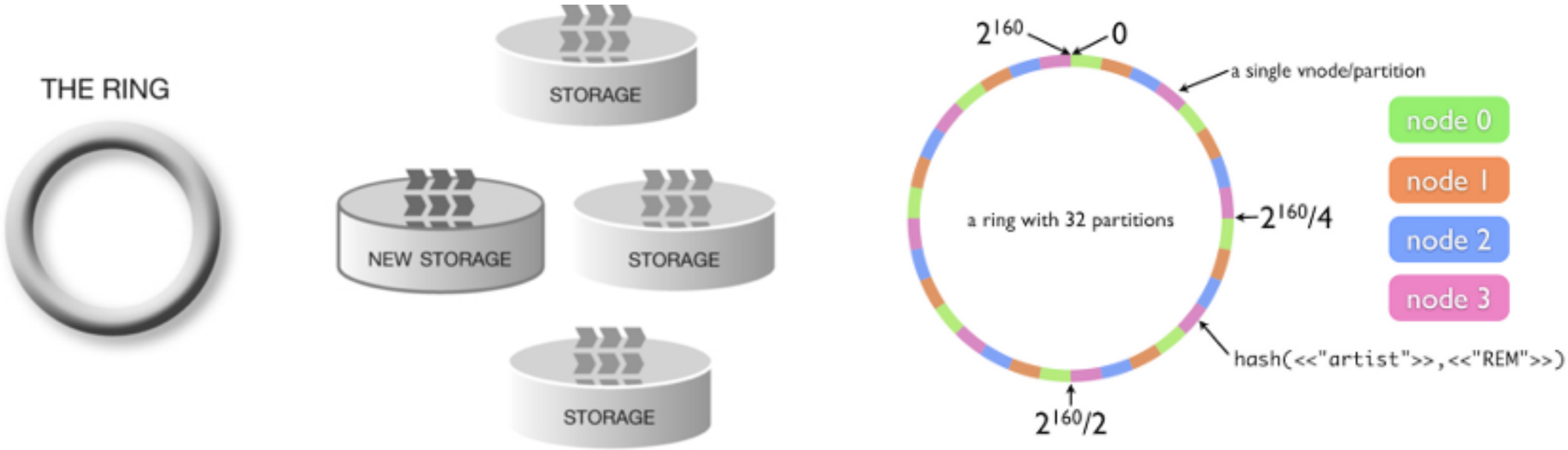
- Objects are protected by storing multiple copies of data so that if one node fails, the data can be retrieved from another node

- That means that you'll always get your data, they will be dispersed on many places, but you could get an old version of them (or no data at all) in some odd cases (like some server overload or failure).

- But there are mechanisms built into Swift to minimize the potential data inconsistency window: they are responsible for data replication and consistency.

Consistent hashing

- **Ring**: represents the space of all possible computed hash values divided in equivalent parts. Each part of this space is called a partition.



- Rings are used to deduce where a particular piece of data is stored.

Data duplication

- By default, Swift stores **3 copies** of every objects, but that's configurable.

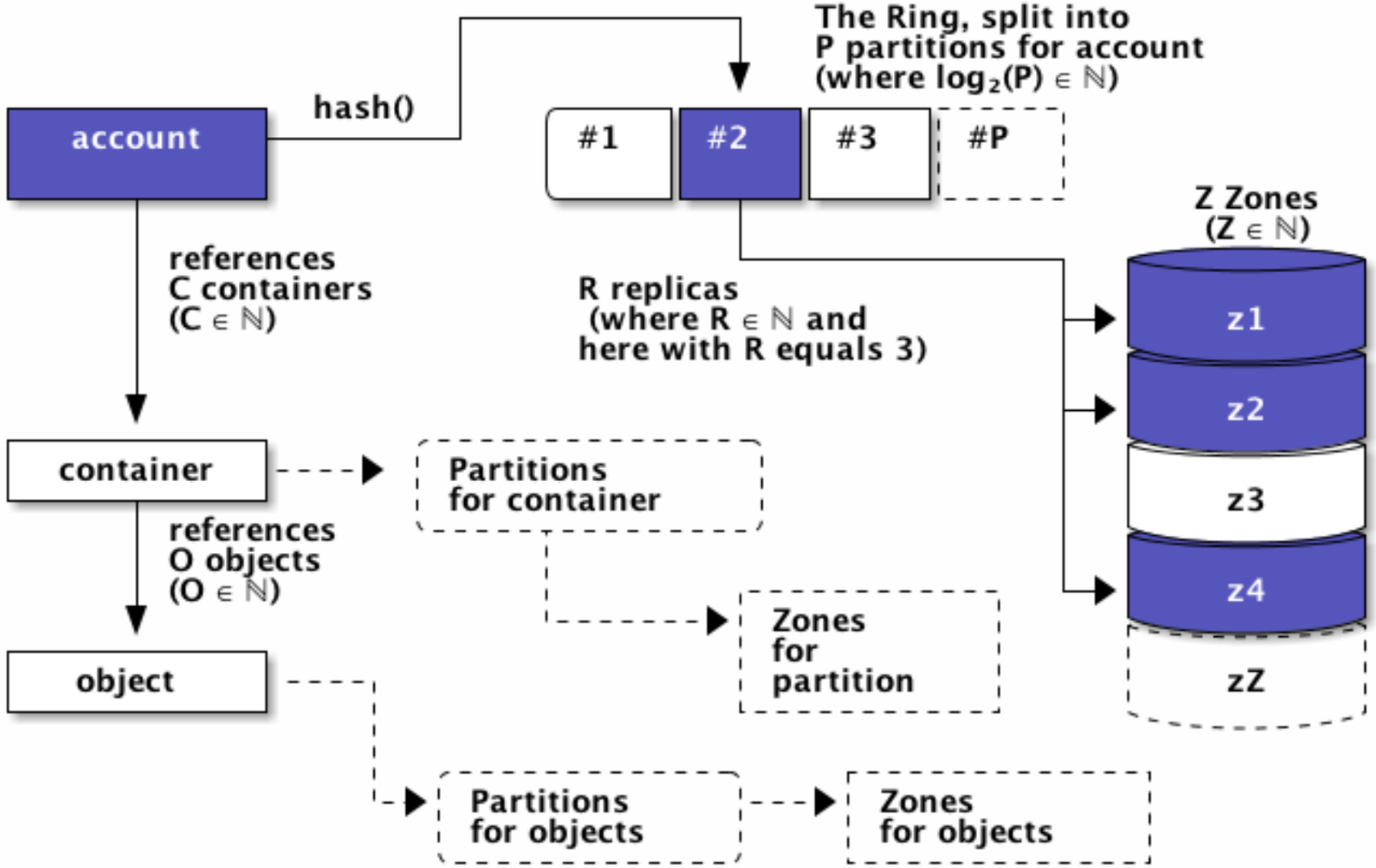
- **Zone**: is an isolated space that does not depend on other zone, so in case of an outage on a zone, the other zones are still available.

- Concretely, a zone is likely to be a disk, a server, or a whole cabinet, depending on the size of your cluster.

- Each partition is replicated; each replica is placed as uniquely as possible across the cluster.

accounts, containers, objects

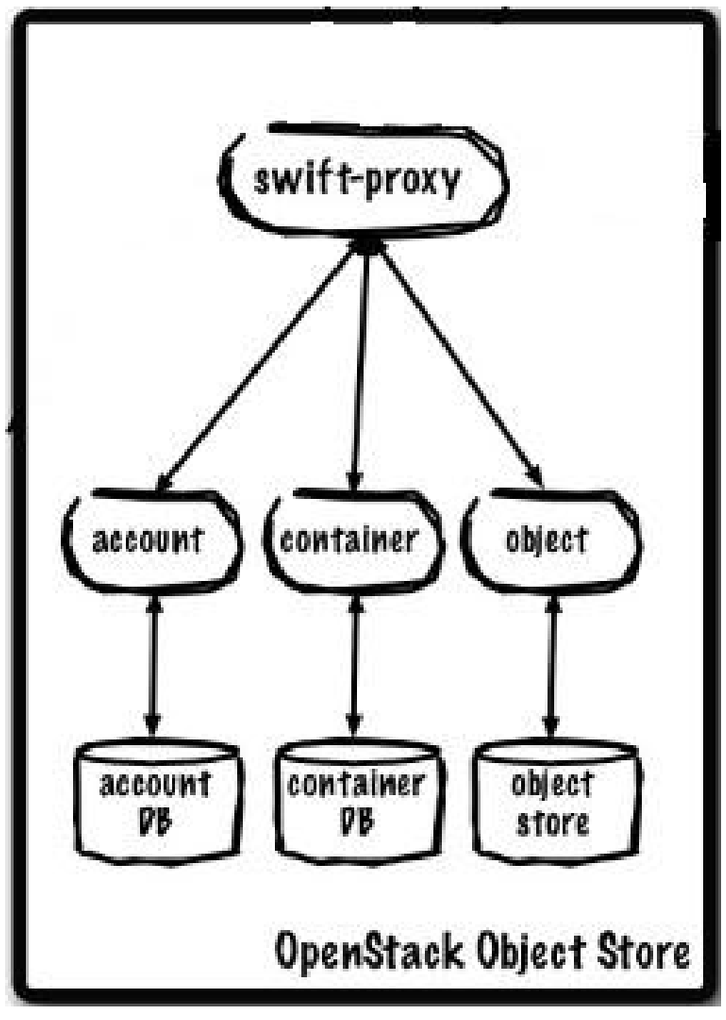
- In Swift, there are 3 categories of thing to store: account, container and objects —> 3 independent rings



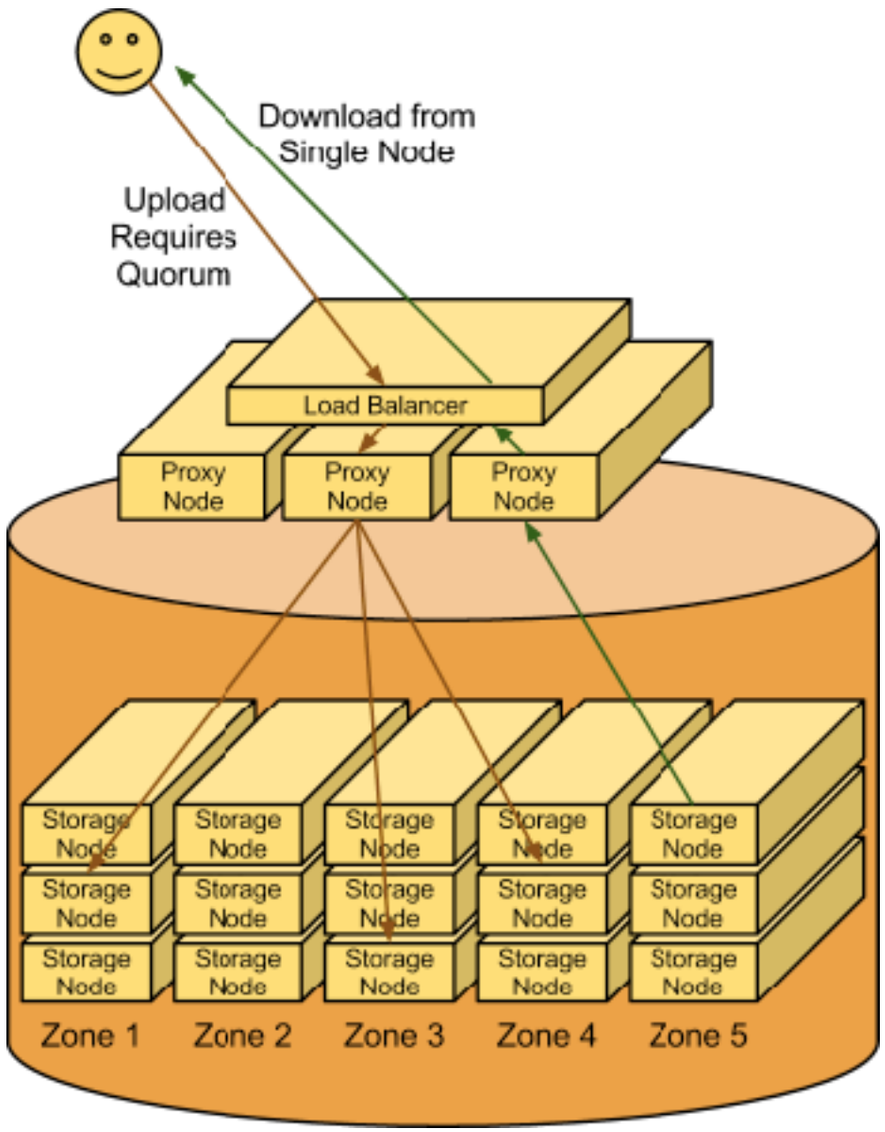
Swift Architecture

Swift Components

- Proxy Server
- Object Server
- Container Server
- Account Server
- Replication
- Updaters
- Auditors
- Reapers



Swift deployment



- Swift is a **two-tier** storage system consisting of
  - a ***proxy tier***, which handles all incoming requests;
  - an ***object storage tier*** where the actual data is stored.
- In addition, **consistency** processes run to ensure the integrity of the data.

Data Access

- By default, Swift OpenStack provides
  - RESTful APIs
  - CLI client (python-swiftclient)
- It is possible to enable standard interfaces, like **S3** (Amazon-compliant APIs) and **CDMI** (Cloud Data Management Interface), adding the corresponding middleware name in the proxy-server pipeline and its parameter section.
  S3 Example: /etc/swift/proxy-server.conf

```
[pipeline:main]
pipeline = healthcheck cache swift3 s3token  authtoken keystoneauth proxy-logging  proxy-server
…
[filter:swift3]
use = egg:swift3#swift3

[filter:s3token]
paste.filter_factory = keystone.middleware.s3_token:filter_factory
auth_port = 35357
auth_host = controller
auth_protocol = https
```

- For CDMI the following extra package has to be installed too: https://github.com/osaddon/cdmi

Data Security

- Protect the cluster endpoint, enabling SSL support in the proxy servers or using an SSL terminator (e.g. Pound)

- Node-to-node communication happens via HTTP —> deploy them on secure network (e.g. VLAN)

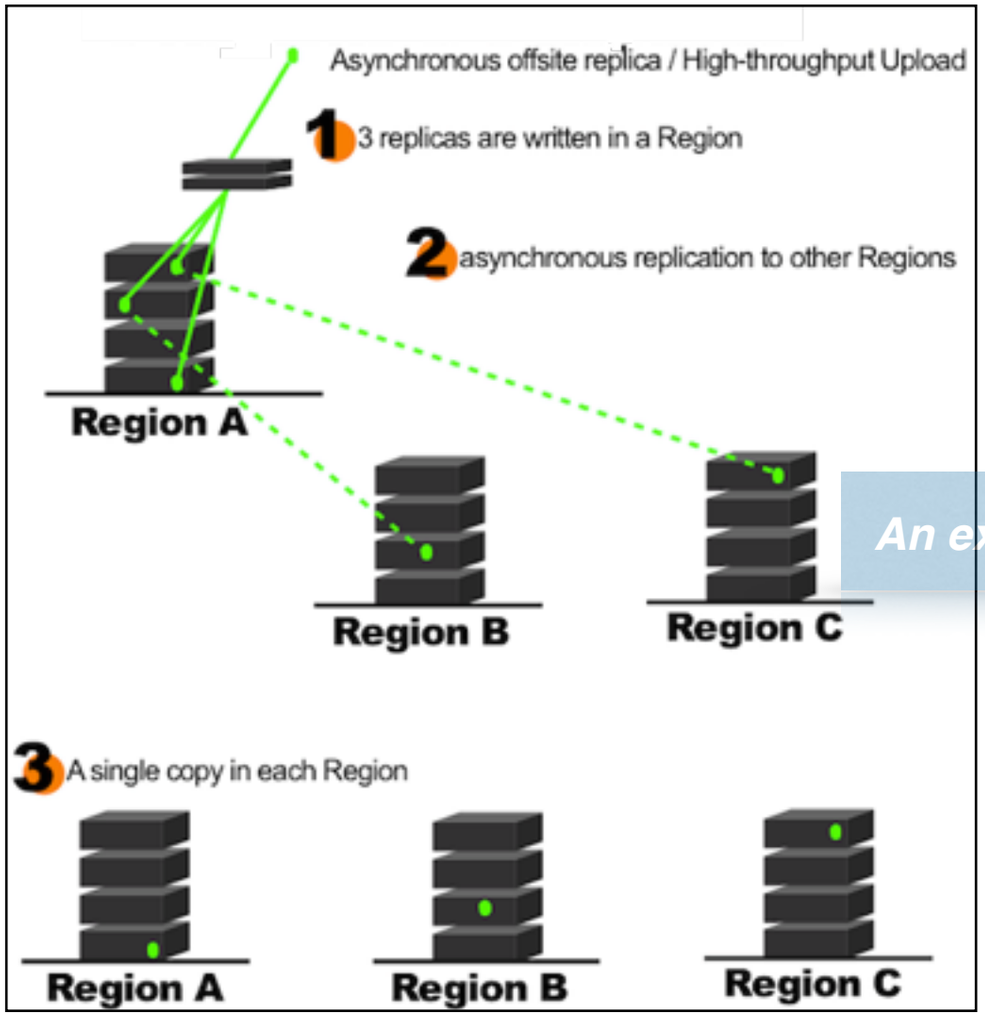
- Node-to-node replication: rsync traffic is not encrypted —> use a secure network

- Data encryption: relies on the backend storage system

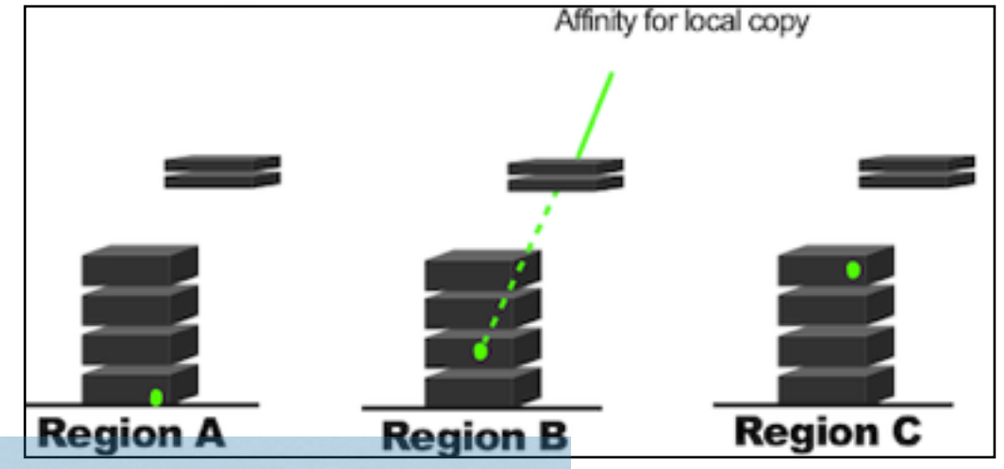
Geographically distributed Swift cluster

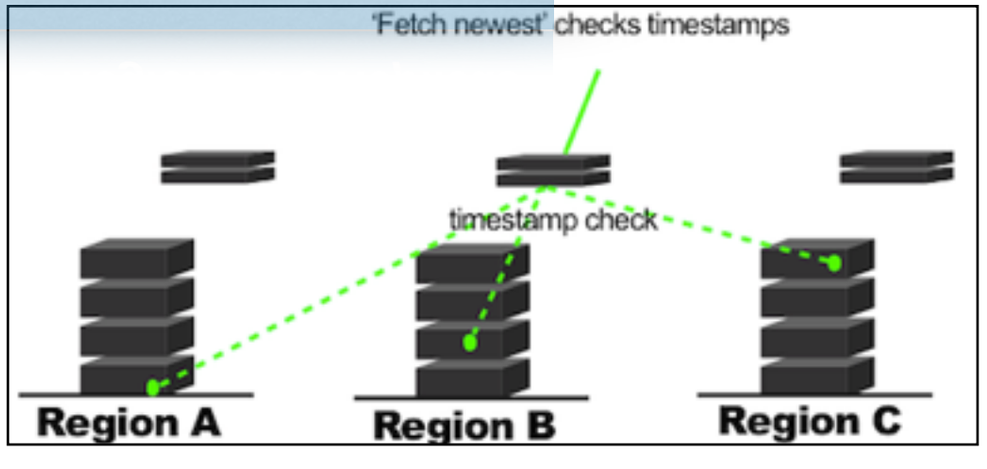
Multi-regional clusters

- A **Region** represents an additional level of tiering, or a group of zones, so all the devices that belong to zones constituting a single region must belong to this region.
- The proxy nodes will have an **affinity** to a Region and be able to optimistically write to storage nodes based on the storage nodes' Region. Optionally, the client will have the option to perform a write or read that goes across Regions (ignoring local affinity),if required.

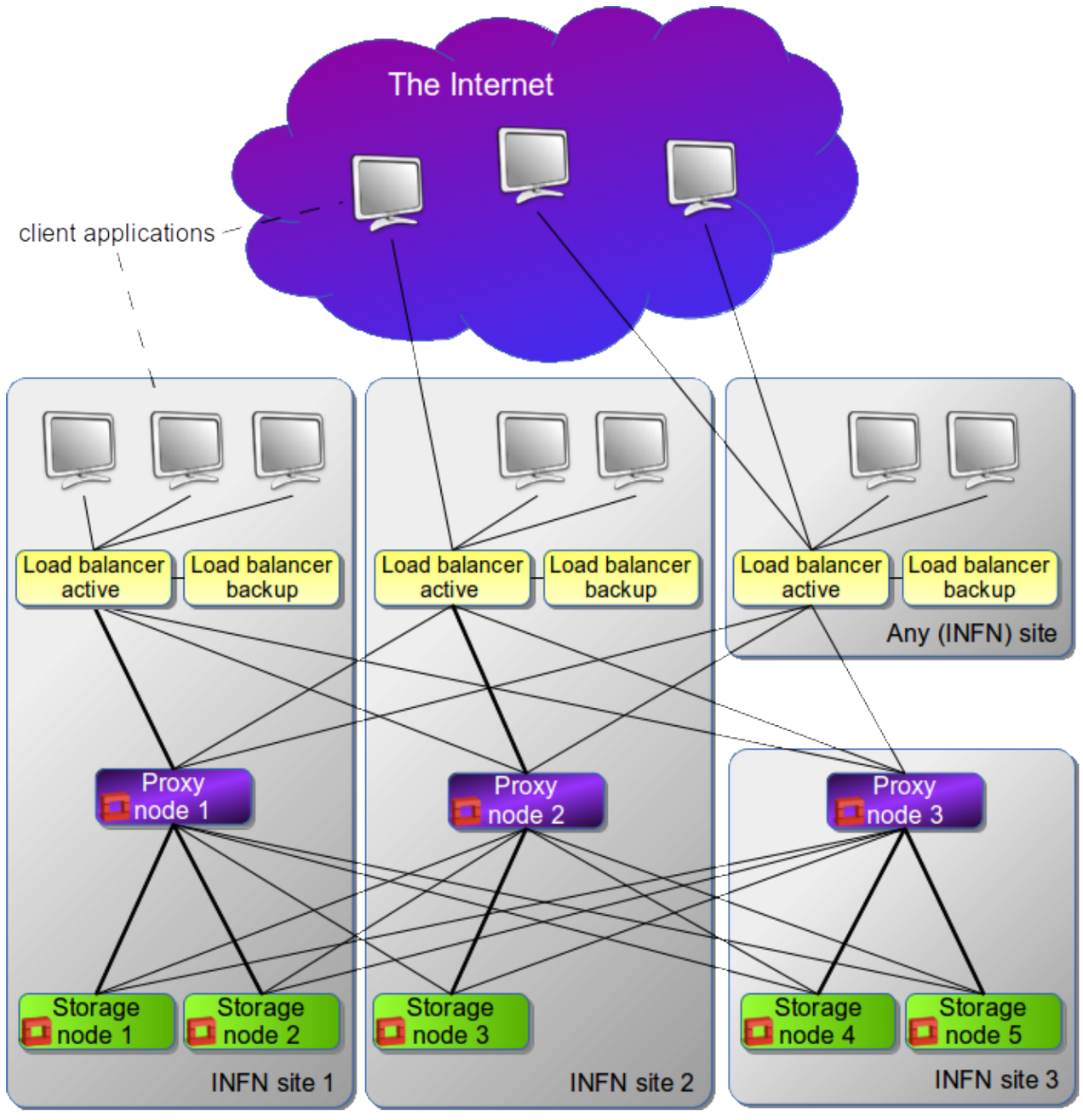


An example: 3 Regions & 3 Replicas





INFN Distributed Set-up



- Installation on 3 different INFN sites: LNGS, BA, PD
- People involved:
  - S. Stalio, M. Panella (LNGS)
  - M. Antonacci (BA)
  - S. Bertocco, C. Aiftimiei (PD)

Swift Monitoring

Hardware failure

- Hard drive failure detection in Swift

  ✓ Use **swift-drive-audit**

  - based on the observation that when a drive is about to fail, error messages will spew into /var/log/kern.log.

  - swift-drive-audit is a script can be run via cron to watch for bad drives. If errors are detected, it will unmount the bad drive, so that Swift can work around it.

Cluster health monitoring

- Use **swift-dispersion-report** tool
  - based on config file /etc/swift/dispersion.conf

```
[dispersion]
auth_url = http://localhost:8080/auth/v1.0
auth_user = test:tester
auth_key = testing
endpoint_type = internalURL
```

```
$ swift-dispersion-report
Queried 2621 containers for dispersion reporting, 19s, 0 retries
100.00% of container copies found (7863 of 7863)
Sample represents 1.00% of the container partition space

Queried 2619 objects for dispersion reporting, 7s, 0 retries
100.00% of object copies found (7857 of 7857)
Sample represents 1.00% of the object partition space
```

Cluster telemetry and monitoring

- **swift-recon** middleware

The Swift Recon middleware can provide general machine stats (load average, socket stats, /proc/meminfo contents, etc.) as well as Swift-specific metrics:

- The MD5 sum of each ring file.
- The most recent object replication time.
- Count of each type of quarantined file: account, container, or object.
- Count of “async_pendings” (deferred container updates) on disk.

Swift-recon

- Config section to be added in object/container/account-server.conf

```
[pipeline:main]
pipeline = recon object-server

[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
```

```
[pipeline:main]
pipeline = recon container-server

[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
```

```
[pipeline:main]
pipeline = recon account-server

[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
```

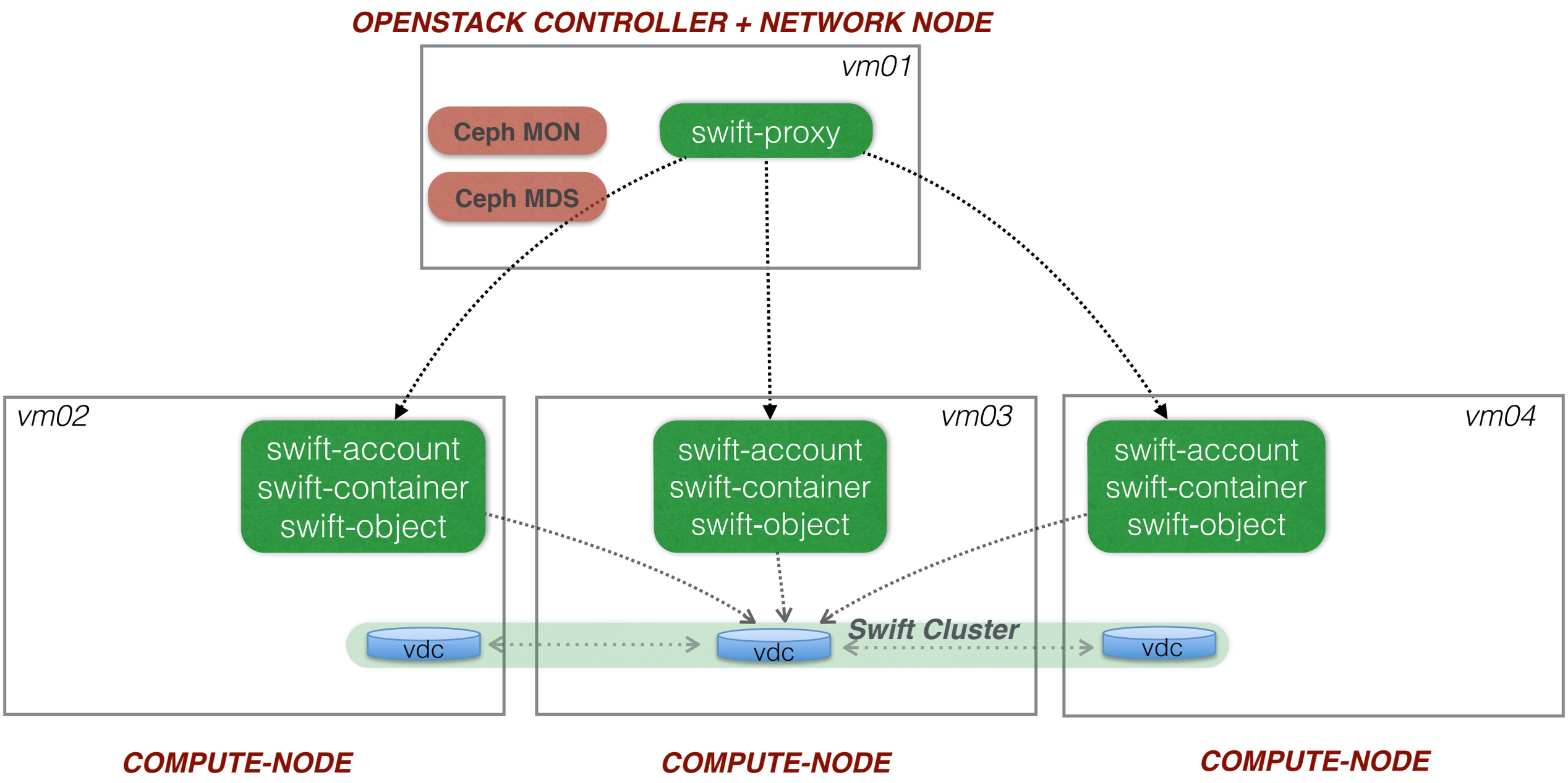
- Following information available

Request URI	Description
/recon/load	returns 1,5, and 15 minute load average
/recon/mem	returns /proc/meminfo
/recon/mounted	returns ALL currently mounted filesystems
/recon/unmounted	returns all unmounted drives if mount_check = True
/recon/diskusage	returns disk utilization for storage devices
/recon/ringmd5	returns object/container/account ring md5sums
/recon/quarantined	returns # of quarantined objects/accounts/containers
/recon/sockstat	returns consumable info from /proc/net/sockstat|6
/recon/devices	returns list of devices and devices dir i.e. /srv/node
/recon/async	returns count of async pending
/recon/replication	returns object replication times (for backward compatibility)
/recon/replication/<type>	returns replication info for given type (account, container, object)
/recon/auditor/<type>	returns auditor stats on last reported scan for given type (account, container, object)

Swift
Installation & Configuration

Swift tutorial

manual installation: https://github.com/infn-bari-school/Swift/wiki



Roadmap

- Create a swift user and the Object Storage endpoint in Keystone
- Install account/container/objects on vm0[2-4]
- Install proxy-server on vm01:
  - ✓ configure the service
  - ✓ create the rings (for each storage device on each node add entries to each ring)
  - ✓ rebalance the rings
  - ✓ disseminate the rings
- Verify installation using the CLI tool and/or Horizon